Carnegie Mellon
Software Engineering Institute

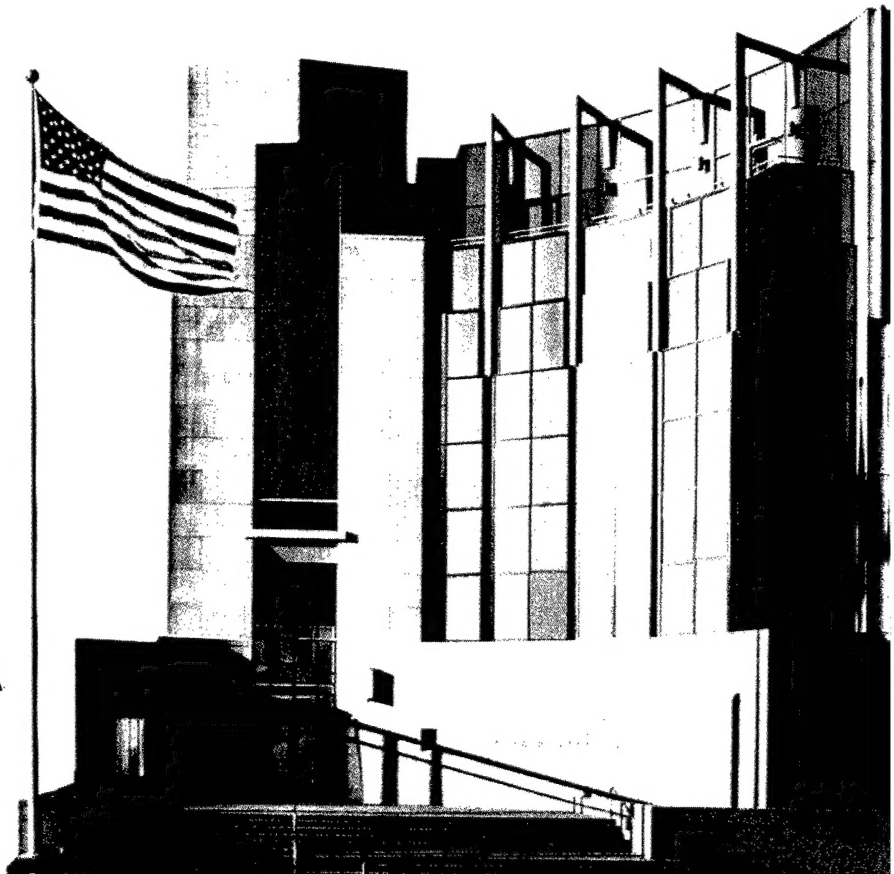
Architecture Reconstruction Case Study

Liam O'Brien
Christoph Stoermer

April 2003

Architecture Tradeoff Analysis Initiative

Technical Note
CMU/SEI-2003-TN-008



Architecture Reconstruction Case Study

Liam O'Brien
Christoph Stoermer

April 2003

Architecture Tradeoff Analysis Initiative

Technical Note
CMU/SEI-2003-TN-008

Contents

List of Figures

List of Tables

Executive Summary

This technical note outlines an architecture reconstruction carried out at the Software Engineering Institute (SEI[SM]) on a software system called VANISH that was developed for prototyping visualizations. There were multiple goals for this reconstruction effort. The first was to understand the architecture of the VANISH system. The second was to use a new architecture reconstruction tool called ARMIN (Architecture Reconstruction and MINing) to support the reconstruction. The third goal was to make sure that this new tool provides at least the same capabilities as the Dali Workbench that was used in a prior reconstruction of the VANISH system [Kazman 97].

During the reconstruction several architectural views were generated through abstraction of low-level information extracted from the system. These views show the components of the system and the relations among them. The ARMIN tool provides the ability to visualize and manipulate the set of views generated during the reconstruction.

The reconstruction shows that the VANISH system is divided into several layers. When the system was originally designed, it was designed in a strictly layered fashion. However, from the views generated in the reconstruction we identified that this strict layering was compromised and interlayer bridging occurs in several places. The reconstruction results can be used to further examine why the relations between the components in the various layers cause the bridging to occur.

During the reconstruction exercise, we found that the same capabilities that were available in the Dali Workbench have been incorporated into the ARMIN tool and that the ARMIN tool provides additional capabilities, including better presentation and layout of the generated architectural views and better capabilities for manipulating them.

[SM] SEI is a service mark of Carnegie Mellon University.

Abstract

This report outlines an architecture reconstruction carried out at the Software Engineering Institute (SEI[SM]) on a software system called VANISH that was developed for prototyping visualizations. The goals of the reconstruction were to understand the existing VANISH system and to use a new architecture reconstruction tool, called ARMIN, for the reconstruction, while ensuring that ARMIN has at least the same capabilities as the Dali Architecture Reconstruction Workbench.

During the reconstruction several architectural views were generated through abstraction of low-level information extracted from the system. These views show the components of the system and the interfaces among them. The ARMIN tool provides the ability to visualize, navigate, and manipulate the set of views generated, and yields results technically compatible with the Dali Workbench but with improved presentation and layout.

1 Introduction

This report outlines an architecture reconstruction carried out on a software system called VANISH that was developed for prototyping visualizations. The VANISH system consists of 50,000 lines of code and is implemented in C++. It was designed in a strictly layered fashion.

There were multiple goals for this reconstruction effort. The first was to generate several architectural views of the VANISH system and to determine if this strict layering is adhered to in the implementation. The second goal was to use a new tool for architecture reconstruction called ARMIN (Architecture Reconstruction and MINing) and to determine the usefulness of this tool in supporting the reconstruction process. The third goal was to make sure that ARMIN has at least the same capabilities for reconstruction as the Dali Architecture Reconstruction Workbench.

Kazman has written a prior case study in which the Dali Workbench was used to reconstruct the VANISH system [Kazman 97], and O'Brien has written a case study showing reconstruction of several other systems [O'Brien 01]. This technical note does not give an in-depth detailed description of the ARMIN tool, nor does it contain a detailed discussion of the differences between the ARMIN tool and the Dali Workbench. This will be the subject of a separate technical report.

The reconstruction process that was followed is shown in Figure 1. This process consists of the following steps:

1. Source information extraction. In this step a set of elements and relations is extracted from the system and loaded into ARMIN.
2. Architectural view composition. In this step views of the system's architecture are generated by abstracting the source information through aggregation and manipulation. The views are presented to the reconstructor in the ARMIN tool, and the user can navigate and manipulate them.

The source code and any documentation are input to the reconstruction process. During the reconstruction process, the reconstruction would use the knowledge of the maintainers and developers in generating the architectural views and they would assist in the process.

The end result of the reconstruction process is a set of architectural views of the VANISH system. These views show various characteristics of the system; for example, that it is decomposed into several layers. These views are presented to the user in the View Generator component of the ARMIN tool, and the user can navigate and manipulate them using the tool. By selecting a particular component or connector between components, the user can see its details.

The View Generator contains an Interpreter that provides the capability of loading and running command scripts to carry out most of the Architectural View Composition step automatically. A command script can be written in an editor and loaded into the tool, and then be used to carry out manipulations on the data in the database and produce new views as a result.

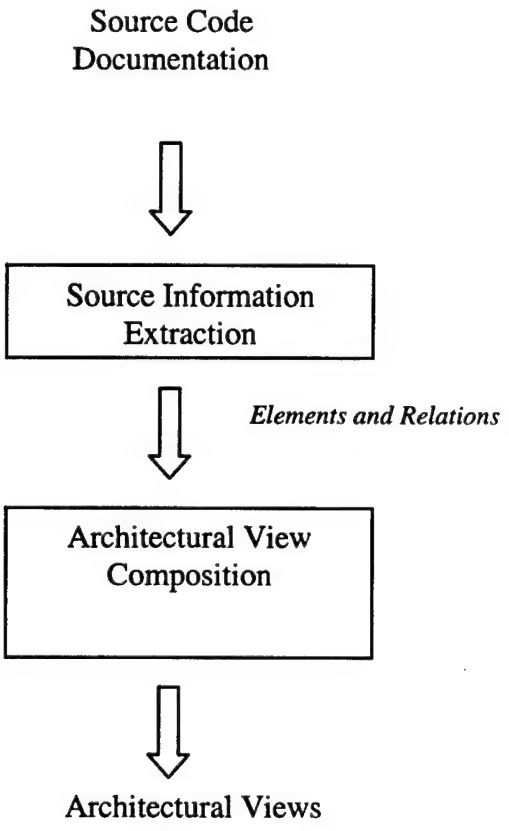


Figure 1: Reconstruction Process

The remainder of this technical note is organized as follows. Chapter 2 outlines the source information extraction. Chapter 3 details the reconstruction process, and Chapter 4 provides a summary, as well as planned additions to the ARMIN tool.

2 Source Information Extraction

The information needed from the VANISH source code had been extracted in the previous reconstruction exercise using the Dali Workbench. In the previous reconstruction, the VANISH source code was analyzed using program analysis tools. Information was extracted from the source code in the form of a set of elements and relations among these elements. Many program analysis tools can do this type of analysis, depending on the language in which the system is implemented. In this case the language is C++ and the set of elements and relations, shown in Table 1, was extracted from the VANISH system and used to assist in the reconstruction of the system.

Element	Relation	Element	Description
Function	Calls	Function	A static function call
Class	defines_fn	Function	Functions defined in classes
Class	has_member	Member_variable	Member variables of a class
Function	defines_var	Local_variable	Local variables of a function
Class	has_subclass	Class	Class hierarchy
Class	has_friend	Class	Friends of a class
File	Contains	Function	Functions defined in files
File	defines_global	Global_variable	Global variables defined in a file
File	Defines	Class	Classes defined in files
File	depends_on	File	Dependencies between files

Table 1: Elements and Relations Extracted from the VANISH System

Most of these analysis tools provide the ability to output their analysis results in a text file that can be manipulated using a scripting language, such as Perl, into the format required for the ARMIN reconstruction tool. One important format used in ARMIN is the Rigi Standard format (RSF) [Müller 93] (a tuple-based data format in the form of "relation <entity1> <entity2>"). ARMIN also supports the Graphical eXtensible Markup Language (GXML).

Other sources of information can be used in addition to analyses of source code. In this case the "Makefile" for the system was analyzed to identify the information for the "depends_on" relation. This relation captures dependencies between source files that contribute to particular executables.

Kazman provides more details about the types of analysis that can be carried out and the types of tools that support them [Kazman 02].

This information was loaded into ARMIN. A view of this information visualized by the tool is shown in Figure 2. This view shows all the elements and relations that have been extracted and loaded into ARMIN.

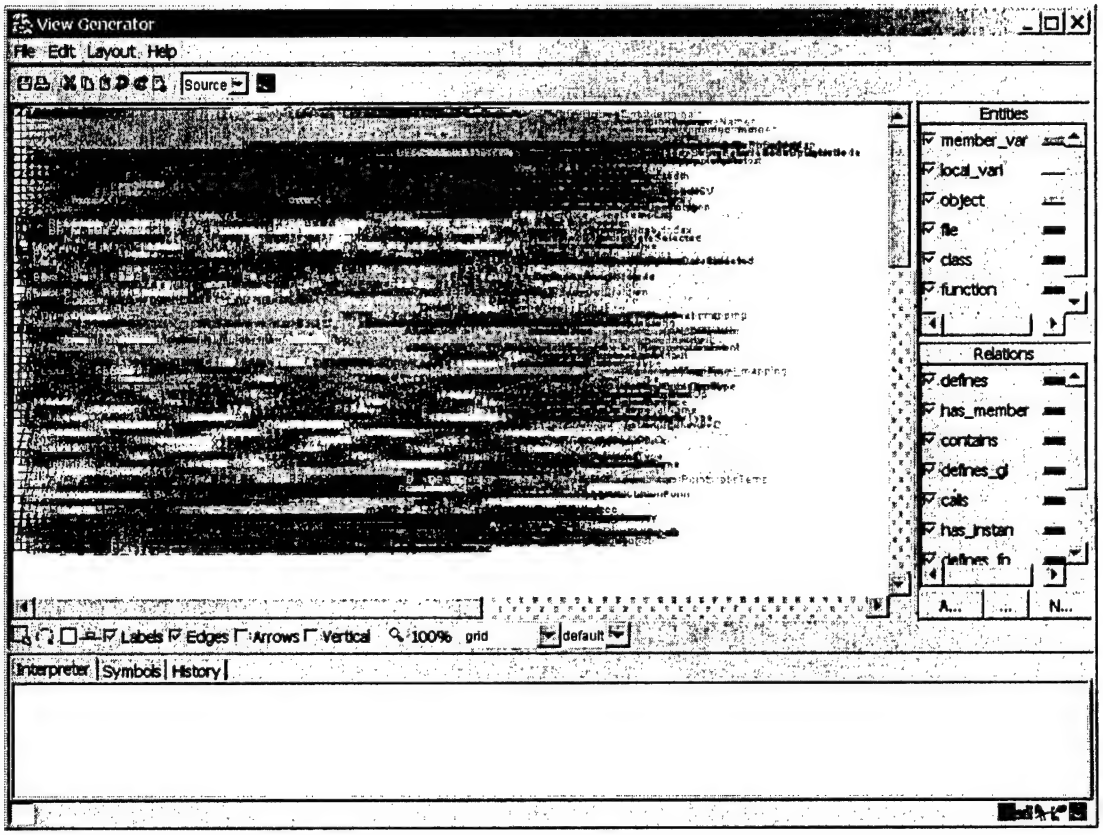


Figure 2: "White-Noise" View Showing All of the Elements and Relations

On the right-hand side of Figure 2, the entity (element) and relation types are listed in a selection panel. It is possible to select the element and relation types to be displayed in the view and to filter out certain elements and relations. This selection panel is available for each view that is generated. In the Dali Workbench, it was possible to filter by navigating the menu system to open two filtering windows, one for elements and another for relations. Also

in ARMIN, colors can be applied dynamically to the different element and relation types at any time. This capability is only available in the Dali Workbench if the colors are set *before* the workbench is run.

The bottom portion of the window shown in Figure 2 provides a command interface to the Interpreter. We can write commands in that window to manipulate and present views, or we can load a command script from an external file that contains a series of commands. More details about the command scripts are in the next section.

ARMIN also allows for different views to be presented in the window. As we generate new views we can select the one we want to be currently displayed in the window. This capability is not available in the Dali Workbench. In Dali, once a view is presented and a query is applied to it, it is not possible to redisplay the previous view without restarting the workbench and applying the set of queries to reproduce it.

In Figure 2 the "Source" view is visible. The View Generator has capabilities for hiding node labels and adding arrows to all edges if necessary. The tool also supports different types of view layouts; in this case the view is laid out in a grid style, and we have the ability to zoom in to particular parts of the graph for closer examination.

3 Architectural View Composition

The reconstruction of any system usually involves several activities for abstracting information that ranges from source-level information (consisting of a set of elements and relations) to higher level views of that information. These activities include

- aggregation
- pattern matching
- analyzing documents to help identify abstractions
- interviewing developers and maintainers to help identify abstractions

In this case the original developers of the system were not available, the system is not currently maintained, and little documentation is available.

We began by carrying out the same abstractions that were carried out in the previous reconstruction of the VANISH system using Dali. First we aggregated local information inside functions to hide details that were not architecturally relevant. In order to do this, we wrote the following command script and executed it within the interpreter:

```
#collapse a function's local_variables
$c = desc(system.types.function);
$c.merge(/ext="+");
collapse($c, /graph="FUNCTION+", /type=system.types.function);
show();
```

This command script takes each function and aggregates its local_variables so that they are removed from the view, and adds a plus sign (+) after each function name. The aggregated function itself is of type function. The resultant view "FUNCTION+" is displayed (using the command "show()") in the tool.

A similar aggregation was carried out on classes within the system. In this case the member variables and functions defined in a class were aggregated within the class. The resultant view, CLASS+, shows each class name followed by a plus sign (+).

```
#collapse member functions and variables inside classes
$d = desc(system.types.class);
$d.merge(/ext="+");
collapse($d, /graph="CLASS+", /type=system.types.class);
show();
```

In generating architectural views we carry out other aggregations including aggregating functions and global variables within files and then aggregating files within classes. These aggregations help to reduce the amount of information that we have to deal with and help in creating the structure of the reconstructed architecture. Having carried out these aggregations, we can now identify components from groups of classes. In the following script we identified the set of elements that belongs to the Dialogue component.

```
#create Dialogue component
$diag = {{{"Dialogue"},
    {"vanish-xforms+","PrimitiveOp++","Mapping++",
     "MappingEditor++","MappingLibrary++","Application++",
     "Renderer++","InputValue++","VEC++","MAT++",
     list("Dbg++$",system.types.class),
     list("Event++$",system.types.class),
     desc("PrimitiveOp++",system.types.has_subclass,/dim=1,/grade=1)}
   }};
$comps = $diag;
```

The Dialogue component consists of the set of files and classes listed, those that end with "Dbg" and "Event" and subclasses of PrimitiveOp.

The command script for creating the Utilities component is below. The Utilities component consists of the class Socket, classes that start with "List" and those classes that end with "Map".

```
#create Utilities component
$util = {{{"Utilities"},
    {"Socket++",list("^List",system.types.class),list("Map++$",system.types.class)}
   }};
$comps.append($util);
```

In this way we identified the set of components and produced an architectural view of the VANISH system showing the layers (set of components) and the relations among them. This view is shown in Figure 3, which also shows the Utilities layer. In Figure 4 the Utilities layer has been removed.

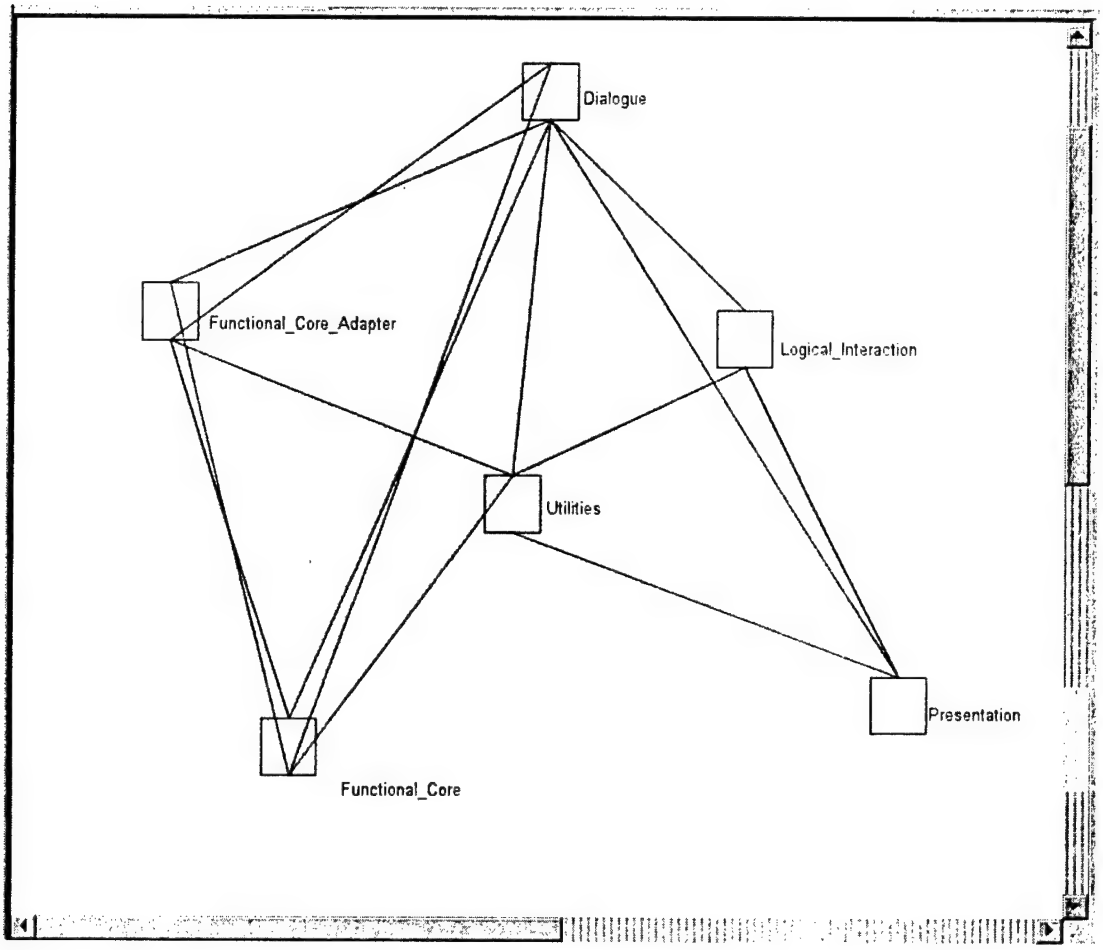


Figure 3: VANISH Architectural View with the Utilities Layer

We can navigate the architectural view that we generated within the View Generator of the ARMIN tool and provide different layouts for the view. We can select a particular node or edge between nodes (in this case, layers and relations) and drill down to identify the low-level elements that make up a layer or the low-level relations that make up an edge.

In the original design, the VANISH system was intended to have strict layering—that is, the Dialogue layer was not to have direct connection to the Functional_Core, and the Functional_Core_Adapter was meant to isolate the functionality of the Functional_Core from Dialogue. However, in the implementation of the system we can identify instances where that strict layering has been compromised, that is, we see instances of interlayer bridging. Figure 4 shows that edges directly connect Functional_Core and Dialogue.

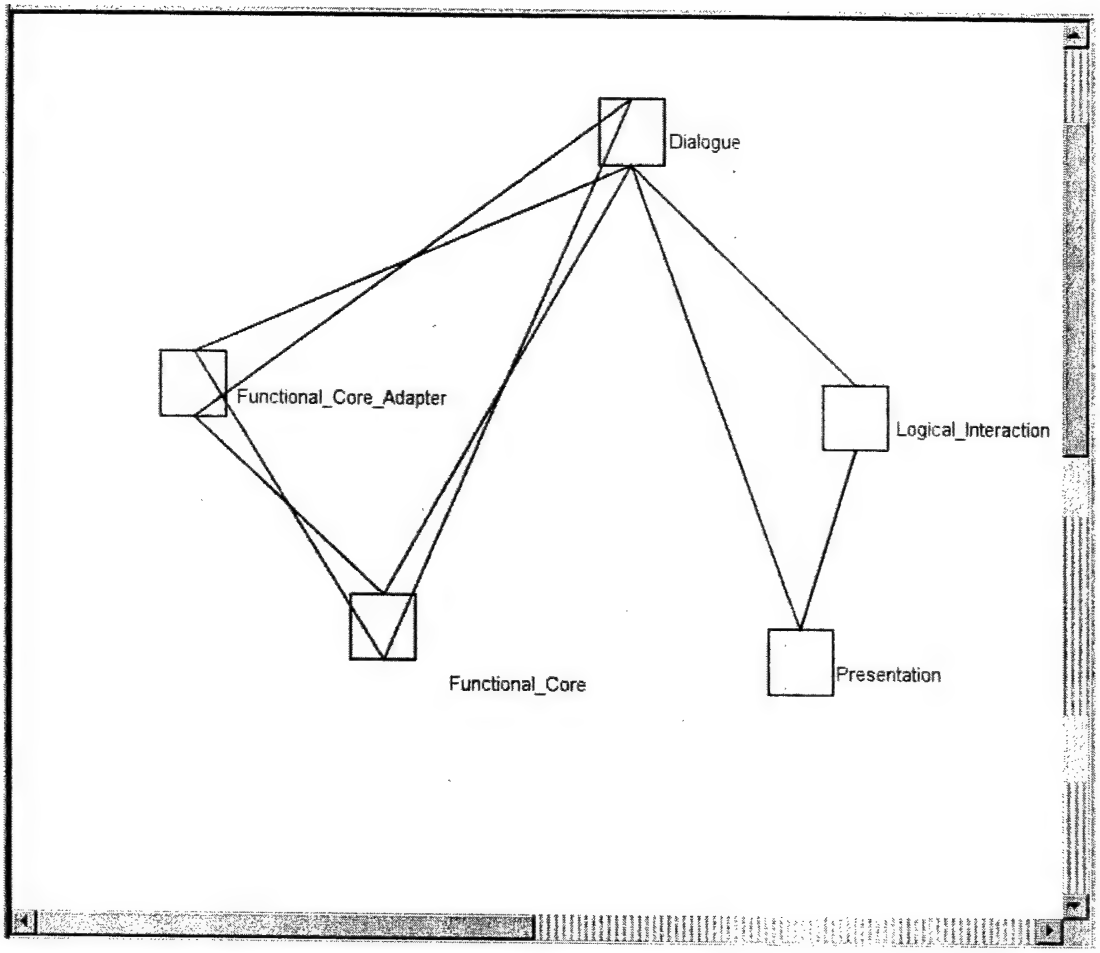


Figure 4: VANISH Architectural View Without the Utilities Layer

To investigate what is causing this inter-layer bridging we can select one of the edges as shown in Figure 5. In this case it is the edge from Functional_Core to Dialogue. This edge means that there is some uses relation between the Functional_Core and Dialogue layers. We can bring up a second window that shows what relations are represented by the edge. At the top of this window, we see that there is a relation between "CallNode++" and "Mapping++". We can navigate to the low-level details to see exactly which set of elements and relations is included in this relation. In this case two function calls are shown in the window (CallNode::MakeNode calls Mapping::Compute and CallNode::MakeNode calls Mapping::Reset), and we can drill down to many other relations that contribute to this edge.

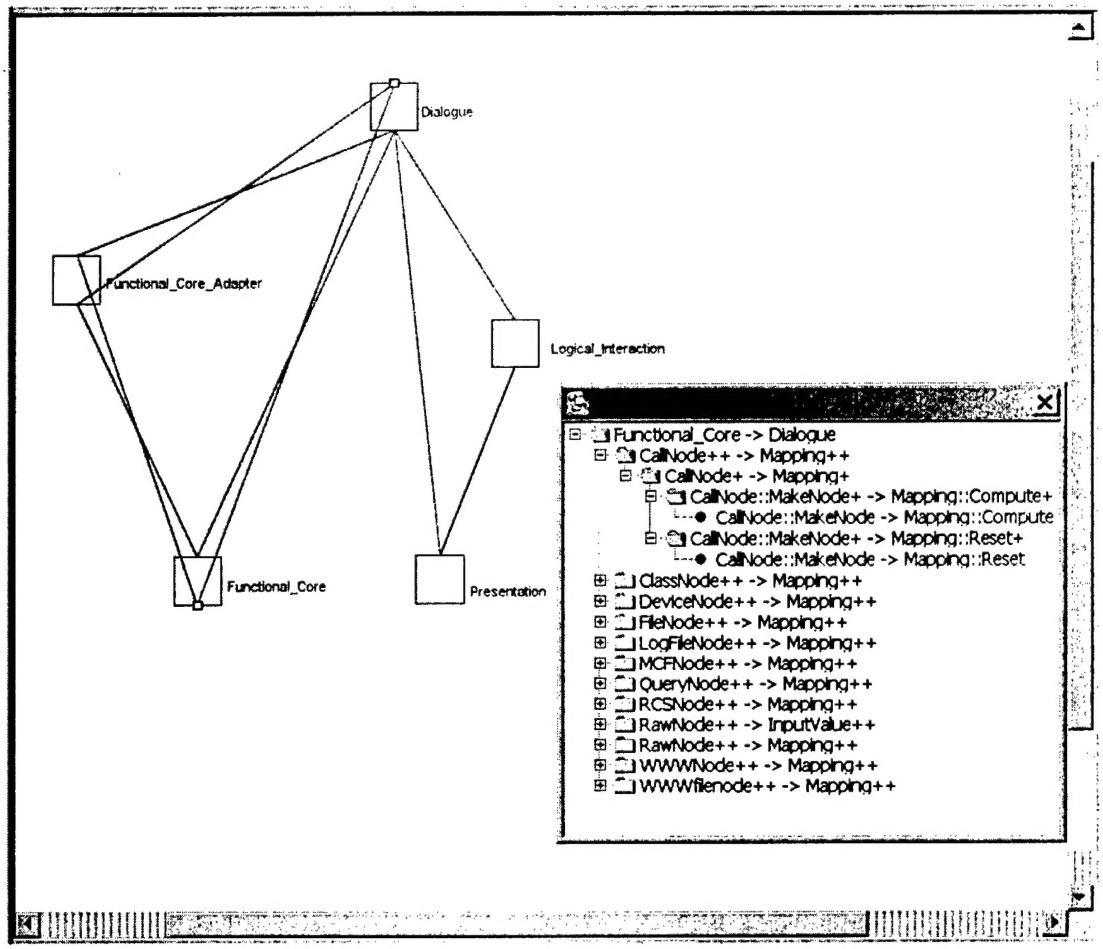


Figure 5: Details of the Relations Between Functional_Core and Dialogue

Using the ARMIN tool gives us the same results as the Dali Workbench. However, the ARMIN tool provides better facilities for displaying and navigating the views that are generated. The interpreter and command scripting provides a better interface for applying queries to the information than was available in the Dali workbench. In the Dali Workbench queries were written in a mixture of SQL and Perl. A new script language has been developed for use in the ARMIN tool.

4 Conclusions

The reconstruction of the VANISH system shows that the system is decomposed into layers, though there is no strict adherence to the use of layers. As a result, interlayer bridging occurs within the implementation. Using the ARMIN tool it is possible to investigate the set of elements and relations between those elements that contributes to that bridging.

Using the ARMIN tool, we generated the same views that had been generated in the previous reconstruction of the VANISH system using the Dali Workbench. The ARMIN tool provides many enhancements to Dali's capabilities including the ability to have different graphical representations for different node types within a view. ARMIN also has the concept of a view stack, which enables users to select and display different views generated during the reconstruction. And ARMIN has the ability to link the information loaded into it to the source code and the ability to be able to navigate to the source code to identify, for example, where a particular function definition occurs in the code (this feature was not shown in this report).

ARMIN also allows for information from multiple systems to be simultaneously loaded into the tool and provides the ability to jump back and forth between the different systems. Also ARMIN's performance is significantly better than that of the Dali Workbench. The Dali Workbench was limited to a particular set of operating systems: Sun Solaris and Linux. The ARMIN tool can run on platforms where the Java Development Kit (JDK) 1.4 is available, which means that it can run on many more platforms. We have used the tool on Windows 2000 and XP, and on various Linux installations.

Further case studies using the ARMIN tool are planned. We are also exploring the addition of further functional capabilities to ARMIN. One of the areas under investigation is the alignment of the views generated by the reconstruction tools with the viewtypes and styles outlined in *Documenting Software Architectures: Views and Beyond* by Clements and associates [Clements 03]. We are investigating what capabilities can be added to ARMIN to support generation of these viewtypes and styles.

References

[Clements 03] Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond.* Boston, MA: Addison-Wesley, 2003.

[Kazman 97] Kazman, R. & Carriere, S. J. *Playing Detective: Reconstructing Software Architecture from Available Evidence*, (CMU/SEI-97-TR-010, ADA330928). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997. <http://www.sei.cmu.edu/publications /documents/97.reports/97tr010/97tr010abstract.html>.

[Kazman 02] Kazman, R.; O'Brien, L.; & Verhoef, C. *Architecture Reconstruction Guidelines, Second Edition* (CMU/SEI-2002-TR-034). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. <http://www.sei.cmu.edu/publications/documents/02.reports /02tr034.html>.

[Müller 93] Müller, H. A.; Mehmet, O. A.; Tilley, S. R.; & Uhl, J. S. "A Reverse Engineering Approach to System Identification." *Journal of Software Maintenance: Research and Practice 5*, 4 (December, 1993): 181-204.

[O'Brien 01] O'Brien, L. *Architecture Reconstruction to Support a Product Line Effort: Case Study* (CMU/SEI-2001-TN-015, ADA395167). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. <http://www.sei.cmu.edu/publications/documents/01.reports /01tn015.html>.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	April 2003	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Architecture Reconstruction Case Study	F19628-00-C-0003

6. AUTHOR(S)

Liam O'Brien, Christoph Stoermer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2003-TN-008

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

This report outlines an architecture reconstruction carried out at the Software Engineering Institute (SEI℠) on a software system called VANISH that was developed for prototyping visualizations. The goals of the reconstruction were to understand the existing VANISH system and to use a new architecture reconstruction tool, called ARMIN, for the reconstruction, while ensuring that ARMIN has at least the same capabilities as the Dali Architecture Reconstruction Workbench.

During the reconstruction several architectural views were generated through abstraction of low-level information extracted from the system. These views show the components of the system and the interfaces among them. The ARMIN tool provides the ability to visualize, navigate, and manipulate the set of views generated, and yields results technically compatible with the Dali Workbench but with improved presentation and layout.

14. SUBJECT TERMS	15. NUMBER OF PAGES
ARMIN, Dali Architecture Reconstruction Workbench, software architecture, software architecture reconstruction, VANISH	24

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL